Technical Note: The Triad of Context-Driven Learning

Most neural networks are described as stacks of layers that gradually transform an input into an output.

An alternative viewpoint is that many neural network architectures can be understood as the interaction of three simple ideas:

- **Context** — determines the computational state.
- **Conditional computation** — determines which computation is performed.
- **Learning** — organizes those computations through backpropagation.

This “Triad” provides a useful way to think about how modern neural networks organize complex computations.

The Three Components

We can describe the three parts using simple notation.

1. Context

C(x)

The context is computed from the current input.

Its job is to answer one simple question:

> **“Where am I in the computational state space?”**

The context might consist of:

- binary decisions,
- routing variables,
- random projection bits,
- activation patterns,
- or any other representation that describes the current situation.

The context does **not** compute the output.

Instead, it determines which computation should be used.

---

2. Conditional Computation

P(C,x)

Once the context is known, the network performs a computation that depends on that context.

The simplest way to think about this is **parameter selection**.

Instead of always using exactly the same parameters, the context selects which parameter blocks participate in the computation.

Conceptually:

```
Input

↓

Compute context

↓

Select parameter blocks

↓

Linear mapping

↓

Next layer
```

Every input may therefore experience a slightly different effective computation.

Composite Linear Mappings

Consider a network with many layers.

Each layer performs a linear mapping.

If the context changes which parameter blocks are active at each layer, then each input follows its own sequence of linear mappings.

The overall computation becomes a **composite linear mapping** assembled specifically for that input.

Different inputs may share many of the same parameter blocks while differing in only a few.

This allows a very large number of possible computations to be built from a much smaller collection of shared parameters.

Why Parameter Sharing Matters

Without parameter sharing, every possible context would require its own completely separate model.

That quickly becomes impractical.

Instead, many different contexts reuse the same parameter blocks.

This creates a **factorization** of the overall problem.

Rather than learning one enormous computation for every possible situation, the network learns many reusable computational pieces.

Those pieces are combined differently depending on the current context.

Factorization is one of the main reasons neural networks can solve highly complex problems without requiring an impossible number of parameters.

The Role of Backpropagation

Backpropagation is often described simply as a way of computing gradients.

A more intuitive view is that backpropagation gradually learns how these shared computational pieces should work together.

For every training example:

- a context is generated,
- a particular collection of parameter blocks is selected,
- the resulting computation produces an output,
- the prediction error is measured,
- backpropagation adjusts the selected parameters.

Over many thousands or millions of training examples, the shared parameters gradually become useful across many different contexts.

Some parameter blocks become broadly useful.

Others become specialized for particular situations.

The network therefore learns both:

- reusable computation,
- specialized computation,

using exactly the same optimization process.

Putting It Together

The three components work together continuously.

```
Input
↓
Context C(x)
↓
Parameter selection
↓
Computation P(C,x)
↓
Prediction
↓
Error
↓
Learning L
↓
Improved shared parameters
```

The next training example repeats the process.

Over time, increasingly effective computations emerge.

Beyond Linear Mappings

Using context to select parameter blocks is only the simplest example.

Exactly the same idea can be extended much further.

The context might instead select:

- an entire small neural network,
- a mixture-of-experts module,
- a transformer block,

- a dynamical model,
- a probabilistic model,
- or another context-driven computation.

The underlying principle does not change.

The context determines **which computation** is performed.

Learning determines **how that computation is organized**.

Connection to Existing Neural Networks

Even conventional neural networks contain elements of conditional computation.

For example, when a ReLU activation becomes zero, the outgoing connections from that neuron no longer contribute to the next layer's computation.

In effect, some computations become active while others become inactive depending on the input.

Architectures that make this routing more explicit simply expose and extend the same underlying idea.

Take Away

The Triad provides a simple way to think about many modern neural network architectures.

- **Context C(x)** determines the computational state.
- **Conditional computation P(C,x)** selects the computation performed for that state.
- **Learning L** uses backpropagation to organize the shared parameters into reusable computational building blocks.

Viewed this way, a neural network is not simply learning one large nonlinear function. It is learning a large collection of shared computational components and how to combine them efficiently for different inputs. That organization—the factorization of computation through context and parameter sharing—is one of the key reasons modern neural networks are both expressive and computationally practical.